PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

MONTE CARLO TREE SEARCH PARA EL PROBLEMA DE CARGA DE CONTENEDORES

CARLOS ALBERTO ALTAMIRANO RAMÍREZ

INFORME FINAL DE PROYECTO DE TÍTULO
PARA OPTAR AL TÍTULO PROFESIONAL DE
INGENIERO DE EJECUCIÓN EN INFORMÁTICA

Diciembre, 2018

PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

MONTE CARLO TREE SEARCH PARA EL PROBLEMA DE CARGA DE CONTENEDORES

CARLOS ALBERTO ALTAMIRANO RAMÍREZ

PROFESOR GUIA: **IGNACIO ARAYA**
PROFESOR CORREFERENTE: **GUILLERMO CABRERA**

INFORME FINAL DE PROYECTO DE TÍTULO
PARA OPTAR AL TÍTULO PROFESIONAL DE
INGENIERO DE EJECUCIÓN EN INFORMÁTICA

Diciembre, 2018

Resumen

Actualmente en la industria de transporte y distribución existen problemas que se les busca dar solución mediante el uso de algoritmos heurísticos de optimización. Uno de estos problemas es el de la Carga de Contenedor Único - CLP por sus siglas en Inglés-, el cual consiste en llenar un contenedor con un conjunto dado de cajas optimizando el volumen total de su capacidad de carga. En este trabajo se propone una solución al problema CLP utilizando una adaptación del algoritmo Monte Carlo Tree Search (MCTS), el cual explora el árbol de búsqueda tomando decisiones de selección de nodo que dependen de un análisis estádistico.

Palabras Clave:Container loading problem, monte carlo tree search, greedy.

Abstract

Currently in the transport and distribution industry there are problems that are looking for solutions through the use of optimization heuristic algorithms. One of these problems is that of the Container Loading Problem - CLP for its acronym -, which consists of filling a container with a given set of boxes optimizing the total volume of its load capacity. In this paper, a solution to the CLP problem is proposed using an adaptation of the Monte Carlo Tree Search (MCTS) algorithm, which explores the search tree by making node selection decisions that depend on a statistical analysis.
Key words: Container loading problem, Monte Carlo tree Search, greedy.

Índice

Lista de Figuras

Lista de Tablas

1. Introducción

El mundo globalizado e interconectado en el que estamos inmersos, es el resultado de grandes procesos transformadores, ideas innovadoras y productos que han llegado a revolucionar las diferentes industrias. Uno de los productos que impactó fuertemente la industria del transporte, fue la creación del contenedor de transporte moderno -Container, su nombre en inglés-, resultado del ingenio de Malcom McLean, su creador, que ya desde la Segunda Guerra Mundial provocó la evolución del transporte marítimo; en ese entonces se utilizaba para transporte seguro de material bélico. Desde entonces, el tamaño de los buques mercantes ha aumentado en tamaño, los puertos se modificaron, se especializaron los equipos de carga, y todo esto para adaptarse a esta nueva forma de traslado de mercancías y satisfacer las demandas globales de transporte. Todo indica que esta última continuará en aumento, dado que según datos oficiales del Banco Mundial, en el año 2000 el tráfico marítimo de contenedores fue de 224.774.536 a nivel global, en contraste con las cifras del año 2017 que indica que se triplicó con 752.704.435 contenedores que circularon al rededor del mundo [24].

El campo de la informática ha tomado éste gran avance como fuente de inspiración para resolver uno de los problemas derivados del uso de contenedores, el *Problema de la Carga de Contenedores*, CLP por sus siglas en inglés (Container Loading Problem). Dicho problema es un problema de optimización combinatoria y pertenece al tipo de problema *Branch-and-Cut*, y su objetivo es maximizar el espacio que se utiliza para cargar el contenedor. Métodos basados en la búsqueda de arboles constructivos que, de manera similar a la heurística constructiva, construyen una o varias soluciones cargando las cajas una a una hasta que no se pueden cargar más cajas dentro del contenedor. Para seleccionar una caja prometedora para colocar a continuación, estos métodos evalúan diferentes alternativas al explorar parcialmente el espacio de búsqueda utilizando una búsqueda de árbol de tamaño limitado que comienza en desde la actual solución parcial [13, 18, 34, 33, 3, 2]. Un método que que utiliza la lógica anteriormente descrita es Beam Search Greedy (BSG) [3, 2], es un algoritmo de búsqueda basada en árboles y reporta algunos de los mejores resultados para CLP hasta la fecha.

Finalmente nosotros proponemos usar una adaptación de la búsqueda de árboles de Monte Carlo (MCTS), que es un método para encontrar decisiones óptimas en un dominio dado tomando muestras aleatorias en el espacio de decisión y construyendo un árbol de búsqueda según los resultados. Ya tuvo un profundo impacto en la inteligencia artificial (IA), en particular juegos y planificación problemas [7].

2. Definición de objetivos

A continuación se definirán los objetivos de esta investigación.

2.1. Objetivo general

Implementar un algoritmo basado en Monte Carlo Tree Search para la resolución del problema de carga de un único contenedor.

2.2. Objetivo especifico

- Comprender el problema de la carga de contenedor único.
- Comprender el método Monte Carlo Tree Search y adaptarlo para resolver el problema.
- Diseñar e implementar el algoritmo Monte Carlo Tree Search utilizando técnicas probabilísticas.
- Realizar experimentos para evaluar el desempeño de la solución propuesta.

3. Estado del arte

El Single Container Loading Problem, es un problema que ha despertado el interes de muchos investigadores, abordandolo con diversas tenicas. Según la teoría de la complejidad computacional, este problema se ubica en la clase *NP-hard* [30], el número exacto de enfoques es reducido, lo que solo permite resolver problemas de tamaño limitado [26, 14, 20, 19]. Las heurísticas convencionales que resuelven CLP incluyen heurísticas constructivas, la que inicia con un contenedor vacío y genera soluciones cargando cajas una por una hasta que no se pueden cargar más cajas; heurísticas de mejora, que intentan mejorar iterativamente soluciones ya conocidas [15, 4, 21, 22]. Por su parte las metaheurísticas incluyen algoritmos genéticos [5, 17]; tabu search [6, 22]; busqueda de vencidad varaible [27]; voraces procedimientos de busqueda adaptativa aleatoria, GRASP, por sus siglas en ingles (greedy randomized adaptive search procedures) [23, 28].

Los métodos recientes más exitosos para SCLP en los casos de prueba estándar han sido enfoques basados en construcción de bloques. Parreño et al. [27] introdujo un algoritmo de espacios maximales que utiliza una búsqueda GRASP en dos fases. El mejor algoritmo estudiado durante este trabajo fue el BSGCLP propuesto por Araya y Riff [3]. El algoritmo utiliza un árbol de búsqueda incompleto (beam search) para construir las soluciones del problema. Las soluciones parciales son evaluadas usando un algoritmo voraz.

Por su parte, el método Monte Carlo que inspira esta investigación, tiene una largo historial dentro de algoritmos numéricos y también ha tenido un éxito significativo en varios algoritmos de IA orientados a juegos, particularmente juegos de información imperfecta como Scrabble y Bridge [16, 31]. Sin embargo, el éxito alcanzado por este algoritmo en la computadora GO[11, 9, 10], programada para jugar el tradicional juego de tablero GO, ha sido el responsable del interés por MCTS, esto se logró mediante la aplicación recursiva de métodos de Monte Carlo durante el proceso de generación de arboles. En los últimos años, MCTS también ha logrado un gran éxito con muchos juegos específicos, juegos en general, en la planificación de tareas complejas del mundo real, problemas de optimización y control, y parece que va a convertirse en una herramienta importante para los investigadores de IA[25, 8, 1, 32, 29]. Se puede proporcionar un agente con cierta capacidad de toma de decisiones con muy poco conocimiento específico de dominio, y su método de muestreo selectivo puede proporcionar ideas sobre cómo otros algoritmos podrían ser hibridados y potencialmente mejorados.

4. Container Loading Problem

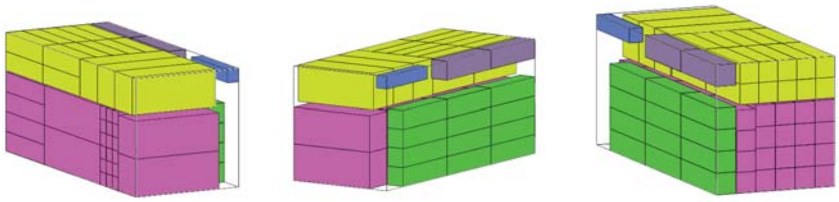
El SCLP -Single Container Loading Problem- es uno de los desafíos del tipo que se enfrenta en la investigación de operaciones. Consiste en llenar un contenedor vacío con un conjunto de cajas dado, con la finalidad de maximizar el volumen total ocupado.

Dado un container de dimensiones L,W y H y un conjunto de cajas

$$C = c1, c2, ..., cN \tag{1}$$

Donde N es la cantidad total de cajas.

El objetivo de CLP es ubicar las cajas del conjunto C dentro del contenedor, maximizando el volumen total que permite cargar. Las caras de las cajas deben ser puestas paralelas a las caras del contenedor y no pueden superponerse con ninguna otra caja como se aprecia en la figura (1).



(a) (b) (c)

Figura 1: Vista del llenado de la matriz considerando bloques de cajas (BSG).

El objetivo puede ser definido como:

$$\sum_{i=1}^{N} p_i * V(c_i) \tag{2}$$

Donde $V(c_i)$ es el volumen de la caja c_i , y p_i es una variable booleana indicando cualquiera de las cajas c_i que han sido ubicadas en el contenedor.

Ahora se debe considerar la orientación de una caja c_i donde sus dimensiones están definidas por longitud l(c_i), ancho w(c_i) y altura c_i tomando como referencia posición original. Una caja puede ser ubicada dentro del contenedor en cualquier orientación. La siguientes restricciones pueden satisfacer todo $i = 1..N$:

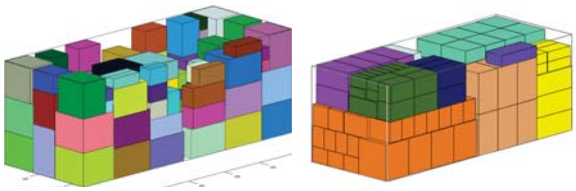
$$
\begin{aligned}
&(\mathrm{l}_i, w_i, h_i) = (l_0(c_i), w_0(c_i), h_0(c_i)) \ \vee \\
&(l_i, w_i, h_i) = (l_0(c_i), h_0(c_i), w_0(c_i)) \ \vee \\
&(l_i, w_i, h_i) = (w_0(c_i), l_0(c_i), h_0(c_i)) \ \vee \\
&(l_i, w_i, h_i) = (w_0(c_i), h_0(c_i), l_0(c_i)) \ \vee \\
&(l_i, w_i, h_i) = (h_0(c_i), l_0(c_i), w_0(c_i)) \ \vee \\
&(l_i, w_i, h_i) = (h_0(c_i), w_0(c_i), l_0(c_i))
\end{aligned}
\tag{3}
$$

También se consideran limitaciones adicionales, tomadas del gran número de restricciones encontradas en la práctica:

- Restricción de orientación: para cada caja, el número de orientaciones permitidas está restringido (por ejemplo, la parte superior de un refrigerador debe estar siempre en la parte superior).
- Restricción de estabilidad (opcional): para garantizar la estabilidad de la carga, los lados inferiores de todas las cajas no colocadas directamente en el piso del contenedor deben estar completamente apoyados por los lados superiores de una o más cajas. Cuando se impone esta restricción, llamamos a la variante de problema resultante: problema de carga de contenedor único con soporte completo (CLP-FS)

4.1. Representación

El llenado del contenedor se hace con un algoritmo que junta dos o más cajas formando bloques, previo a la construcción de la solución. Se realiza de este modo con la finalidad de reducir la complejidad del problema, mejorando la eficiencia de los algoritmos de construcción de soluciones. En la figura 2 se evidencia la ventaja del llenado por bloques (figura 2a) en contraste del llenado por cajas (figura 2b).



(a) (b)

Figura 2: Cajas: 75.06 % (101 cajas colocadas) | Bloques: 89.2 % (12 bloques colocados).

4.2. Función de evaluación *VCS*

Se usa una función heurística que intenta evaluar que tan bueno sería colocar cada uno de los bloques en el espacio seleccionado [2]

4.3. Algoritmo voraz *greedy*

Se explora el espacio de soluciones usando un árbol de búsqueda, filtrando nodos poco prometedores.

1. Se crea un estado que representa el contenedor vacío y se coloca en un conjunto de estados S
2. Para cada estado s en S:
   a. Se generan w estados. Cada estado se genera agregando al estado s uno de los w mejores bloques (según función heurística)
   b. Se genera una solución a partir de cada estado usando un greedy. Los estados son evaluados de acuerdo a la solución obtenida por el greedy
3. De todos los estados generados se seleccionan los w mejores y se colocan en S.

5. Monte Carlo Tree Search para juegos

El algoritmo construye progresivamente un árbol de busqueda parcial, guiado por los resultados de la exploración previa de ese árbol. El árbol se utiliza para estimar los valores de movimientos, con esas estimaciones (en particular con la de los movimientos más prometedores) va convirtiéndose más preciso el árbol que se construye. Cada nodo del árbol de búsqueda representa un estado del dominio y vínculos dirigidos a los nodos hijos conducentes a posteriores estados.[7] Se aplican cuatro pasos por iteración de búsqueda [12], ver figura (3).

5.1. Selección del nodo

A partir del nodo raíz (estado inicial), se aplica recursivamente una directiva de selección secundaria para descender por del árbol hasta que se alcance el nodo expandible más urgente. Un nodo es expandible si representa un estado no terminal y tiene hijos no visitados, es decir, sin expandir (ver *Selection* de la figura 3).

5.2. Expansión del nodo

Se expande el nodo que ha sido seleccionado según la política del árbol (tree policy). Se agregan uno (o más) nodos secundarios a expandir el árbol, de acuerdo con las acciones disponibles. Un nodo es expandible si representa un estado no terminal y no ha visitado,es decir, no se ha expandido, su hijo (ver *Expansion* de la figura 3).

5.3. Simulación

Se lanza una simulación desde el nuevo nodo (s) de acuerdo con la política por defecto para producir un resultado (ver *Simulation* de la figura 3).

5.4. Actualización de información (backpropagation)

El resultado de la simulación es "backed up", es decir, retropropagado, a través de los nodos seleccionados para actualizar sus valores estadísticos. (ver *Backpropagation* de la figura 3).

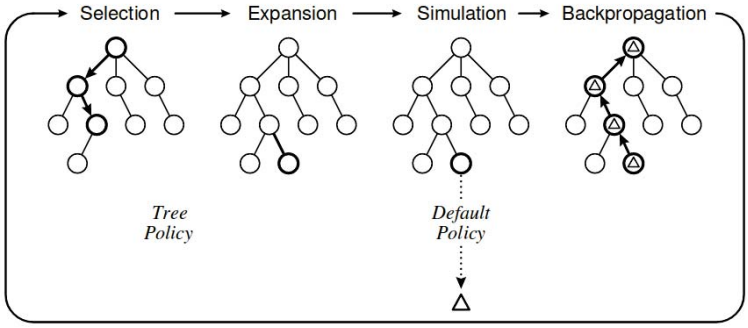


Figura 3: Ejemplo de una iteración general de MCTS

5.5. Problemas y motivación

La investigación de juegos de inteligencia artificial se centra en la suma cero. Juegos con dos jugadores, turnos alternos, espacios de acción discretos, transiciones de estados deterministas e información perfecta. Mientras que MCTS se ha aplicado extensivamente a tales juegos, también se ha aplicado a otros dominios, como juegos para un jugador y problemas de planificación, juegos para múltiples jugadores, juegos en tiempo real y juegos con incertidumbre o movimientos simultáneos. .

6. Propuesta: MCTS para problemas de optimización

Como ya se ha podido comprobar que algoritmos heurísticos usando Beam Search, han dado buenos resultados para CLP [3, 2], nosotros hemos propuesto una nueva adaptación a este algoritmo, en la que hemos incorporado BSG añadiendo el metodo Monte Carlo Tree Search el cual se puede comprender en tres procedimientos principales, que se detallarán a continuación.

6.1. Exploración del árbol de búsqueda

El procedimiento de expansión genera w sucesores para un estado dado s . Primero, se elige un cuboide r de espacio libre . Este cuboide minimiza la distancia a las esquinas del contenedor. Luego, cada sucesor de s se obtiene colocando uno de los w bloques más prometedores en la esquina de r que está más cerca de una esquina del contenedor (*the anchor corner*). Estos bloques se seleccionan de acuerdo con una función heurística basada en el volumen. (ver figura 4)

6.2. Simulaciones y actualización del estado

El procedimiento *simulate* lanza greedies aleatorios que retornan evaluación del objetivo alcanzado. Actualiza valores media, desviación estándar y el conjunto ordenado de hijos. Asocia el hijo correspondiente al estado.

- s': clon del estado s
- n: sorted set of children
- Q: Cola con prioridad de estados seleccionables (priorizada por Probabilidad de generar simulaciones mejores a la mejor encontrada hasta ahora)
- S_0: Estado inicial del problema (contendor vacío)
- children: nodo hijo del estado s

```
MCTS(s0);
push(Q, s0);
while Q is not empty do
    s ← top(Q);
    s' ← simulate(s);
    if s' = NULL then
        pop(Q);
    else
        if s has three children then
            simulate ( children(s) );
            push(Q,children(s));
        else
            if s has more than three children then
                simulate(s');
                push(Q,s');
            else
                default action
            end
        end
    end
end
```

Algorithm 1: Simulación y actualización del estado

6.3. Selección del mejor nodo (promise)

Para seleccionar el mejor nodo se utiliza el procedimiento *calculate promise* que indica la probabilidad de generar simulaciones mejores a la mejor obtenida ($best\_value$) de acuerdo a la media ($mean$) y desviación estandar($dist$); asumiendo que las simulaciones se distribuyen normalmente, se aplica el calculo de $dist$ del nodo, luego se actualiza *promise* con la función de distribución acumulada (cdf) tomando como parámetro $dist$ y $best\_value$.

- $mean$: media calculada con los valores estadísticos de los hijos.
- var: varianza del nodo.
- $dist$: Desviación estándar calculada con la media y la raíz cuadrada de la varianza.
- $best\_value$: mejor valor obtenido de ese estado.

- *cdf* : función de distribución acumulada entre desviación estándar y *best_value*
- *promise* : valor de probabilidad que indica que tan conveniente es el nodo en el que se esta posicionado. Dicho valor se calcula con la función de probabilidad acumulada.

best_value;
dist <– calculate standard deviation ;
cdf <– calculate Cumulative distribution function;
promise <– update promise with *cdf*;

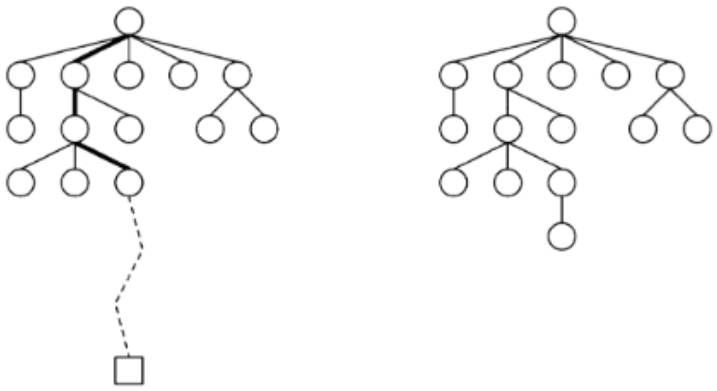


Figura 4: Simulación de un nodo con MCTS

7. Experimentos

Los experimentos se realizaron en un servidor PowerEdge T420, con 2 quad- procesadores Intel Xeon, 2.20 GHz y 8 GB de RAM, corriendo Ubuntu Linux. Los códigos se implementaron en C++ y compilado con gcc.

Para evaluar la efectividad de las heurísticas planteadas se hará uso de una serie de 16 conjuntos de problemas (BR0-BR15) utilizados por Franslau y Bortfeldt para analizar un algoritmo propio. Cada conjunto se compone de 100 casos. Los 16 conjuntos de instancias se pueden clasificar en tres categorías: BR0 consta de sólo un tipo de caja (homogénea); BR1-7 consta de unos pocos tipos de cajas (débilmente heterogénea); y BR8-BR15 consta de hasta 100 tipos de cajas (muy heterogénea). Todas las instancias de prueba imponen una serie de restricciones sobre las posibles orientaciones para las cajas individuales. No se imponen restricciones adicionales.

7.1. Parámetro *max_ node*

Como primera instancia se realizó un set de datos, para ajustar los parámetros del algoritmo y lanzar los experimentos con los mejores valores obtenidos de ellos. Estos resultados se pueden ver en el Anexo A y Anexo B, los que corresponden a los valores de los parámetros de *random* y *eps* respectivamente. *max_node* indica la cantidad máxima de nodos que se pueden tener almacenados en Q, la cola con prioridad. Este experimento se ejecutó 60 segundos por instancia BR. 1

Instances	BSG	MCTS			MCTS_CLP			
			100	200	300	400	500	50
BR1	95.72	93.96	94.0496	94.1318	94.1295	94.1231	94.1228	93.9755
BR2	96.26	94.71	94.8993	94.9454	94.9711	94.9943	94.9952	94.8
BR3	96.51	95.27	95.2853	95.3174	95.3271	95.3493	95.354	95.2142
BR4	96.46	95.37	95.3254	95.3911	95.4154	95.4236	95.4314	95.2681
BR5	96.42	95.26	95.2789	95.3226	95.3317	95.3323	95.3411	95.2331
BR6	96.34	95.14	95.3573	95.3684	95.3903	95.3867	95.3885	95.3051
BR7	96.05	94.78	95.0719	95.0982	95.1037	95.1065	95.1013	95.0291
BR8	95.51	94.35	94.8436	94.8638	94.8649	94.8684	94.8721	94.8051
BR9	95.33	94.27	94.5901	94.6205	94.621	94.6245	94.6278	94.5542
BR10	95.22	94.03	94.5577	94.5699	94.5787	94.5861	94.5813	94.5399
BR11	95.14	93.85	94.4231	94.4275	94.4259	94.4266	94.426	94.3644
BR12	95.01	93.75	94.2931	94.3067	94.3071	94.3083	94.308	94.2623
BR13	94.81	93.54	94.1617	94.1635	94.1684	94.1693	94.1712	94.148
BR14	94.8	93.43	94.0381	94.0437	94.0426	94.0484	94.046	94.0313
BR15	94.64	93.27	93.9427	93.9525	93.9511	93.953	93.9595	93.93
Av. 1-7	96.25	94.93	95.04	95.08	95.1	95.1	95.1	94.98
Av. 8-15	95.06	93.81	94.36	94.37	94.37	94.37	94.37	94.33
Av. 1-15	95.61	94.33	94.67	94.7	94.71	94.71	94.72	94.63

Tabla 1: Resultados con el párametro *max_ node*.

Como se aprecia en la imagen comparativamente BSG alcanza mejores resultados en todas las instancias, a pesar de que se obtienen significativamente buenos resultados.

8. Conclusión

Este estudio propuso un Monte Carlo Tree Search, adaptado para resolver el Container Loading Problem. El enfoque planteado propone una no retropropagación de los valores estadísticos de los nodos, que se utilizan para la selección del nodo a explorar. Además empleó un método propio para calcular ese valor estadístico que indica lo prometedor de cada nodo para ser explorado.

La propuesta puede resolver el problema de maximizar el volumen ocupado de un contenedor en un tiempo razonable. Se puede implementar el método *backpropagation* para investigaciones futuras para acercarse a la estructura esencial de Monte Carlo, puesto que dicho procedimiento se contempló como una actualización de valores de forma local, y es probable que su implementación ayude la toma de decisión del árbol.

Appendices

Anexo A: Resultados obtenidos con parámetro random

Instances	BSG	MCTS	random									
			0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
BR1	95.72	93.96	94.03	93.62	93.47	93.53	93.8335	93.7627	93.7377	93.5934	93.4743	93.5562
BR2	96.26	94.71	94.77	94.75	94.68	94.81	94.84	94.7154	94.7303	94.7751	94.7223	94.6738
BR3	96.51	95.27	95.17	95.27	95.4	95.34	95.3469	95.3669	95.3362	95.3197	95.2053	95.1239
BR4	96.46	95.37	95.27	95.42	95.36	95.32	95.3817	95.3554	95.2788	95.2059	95.2537	95.1946
BR5	96.42	95.26	95.22	95.41	95.47	95.36	95.3239	95.3573	95.2285	95.1514	95.2335	95.0893
BR6	96.34	95.14	95.31	95.5	95.45	95.36	95.3165	95.1976	95.0939	95.1122	95.0557	94.9697
BR7	96.05	94.78	95.03	95.16	95.08	95.06	94.9209	94.8799	94.8101	94.6966	94.6143	94.5429
BR8	95.51	94.35	94.73	94.73	94.74	94.67	94.6285	94.5323	94.5181	94.4633	94.446	94.2741
BR9	95.33	94.27	94.56	94.62	94.47	94.44	94.3884	94.2721	94.152	94.1604	94.058	93.9367
BR10	95.22	94.03	94.41	94.49	94.35	94.27	94.276	94.0729	93.9555	93.9527	93.8382	93.855
BR11	95.14	93.85	94.27	94.3	94.24	94.1	93.9286	93.9031	93.8761	93.7555	93.6065	93.6091
BR12	95.01	93.75	94.26	94.18	94.03	94.01	93.8477	93.7695	93.6912	93.5262	93.4871	93.4955
BR13	94.81	93.54	93.98	94	93.87	93.76	93.6343	93.5656	93.5613	93.3888	93.3209	93.2521
BR14	94.8	93.43	93.89	93.84	93.76	93.65	93.5764	93.4981	93.3847	93.2781	93.2265	93.151
BR15	94.64	93.27	93.87	93.71	93.6	93.49	93.4262	93.2709	93.2554	93.1253	93.1021	92.9903
Av. 1-7	96.25	94.93	94.97	95.02	94.99	94.97	94.9942	94.9478	94.8879	94.8363	94.7941	94.735
Av. 8-15	95.06	93.81	94.25	94.23	94.13	94.05	93.9632	93.8605	93.799	93.706	93.635	93.570
Av. 1-15	95.61	94.33	94.58	94.6	94.53	94.48	94.444	94.3679	94.3073	94.233	94.176	94.114

Tabla 2: Resultados con el parámetro *random*.

Anexo B: Resultados obtenidos con parámetro eps

Instances	BSG	MCTS	eps					
			0,0001	0,0005	0,001	0,005	0,01	0,05
BR1	95.72	93.96	94.03	94.05	94.06	94.24	94.2246	94.0054
BR2	96.26	94.71	94.77	94.89	94.91	94.89	94.8	94.6251
BR3	96.51	95.27	95.17	95.25	95.26	95.25	95.1099	94.6274
BR4	96.46	95.37	95.27	95.28	95.29	95.16	95.0898	94.3896
BR5	96.42	95.26	95.22	95.24	95.22	95.13	95.0259	94.4555
BR6	96.34	95.14	95.31	95.27	95.28	95.07	95.0526	94.3132
BR7	96.05	94.78	95.03	95	94.94	94.75	94.6279	93.8105
BR8	95.51	94.35	94.73	94.75	94.66	94.48	94.3606	93.5997
BR9	95.33	94.27	94.56	94.52	94.49	94.36	94.2428	93.3787
BR10	95.22	94.03	94.43	94.41	94.36	94.17	94.0474	93.2357
BR11	95.14	93.85	94.27	94.32	94.25	94.07	93.9749	93.2839
BR12	95.01	93.75	94.27	94.17	94.16	93.98	93.8238	93.0092
BR13	94.81	93.54	93.98	93.99	93.93	93.78	93.6751	92.8237
BR14	94.8	93.43	93.9	93.9	93.89	93.66	93.5804	92.7589
BR15	94.64	93.27	93.88	93.84	93.81	93.56	93.4369	92.6389
Av. 1-7	96.25	94.93	94.97	95	94.99	94.93	94.8476	94.3181
Av. 8-15	95.06	93.81	94.25	94.24	94.2	94.01	93.8927	93.0910
Av. 1-15	95.61	94.33	94.59	94.59	94.57	94.44	94.3383	93.6636

Tabla 3: Resultados con el parámetro *eps*.

Referencias

[1] F. Teytaud A. Rimmel and T. Cazenave. Optimization of the nested monte-carlo algorithm on the traveling salesman problem with time windows. *Proc. Applicat. Evol. Comput. 2*, pages 501–510, 2011.

[2] Ignacio Araya, Keitel Guerrero, and Eduardo Nuñez. Vcs: A new heuristic function for selecting boxes in the single container loading problem. *Computers & Operations Research*, 82:27–35, 2017.

[3] Ignacio Araya and M-C Riff. A beam search approach to the container loading problem. *Computers & Operations Research*, 43:100–107, 2014.

[4] EE Bischoff, F Janetz, and MSW Ratcliff. Loading pallets with non-identical items. *Eur J Oper Res*, 84(3):681–92, 1995.

[5] Andreas Bortfeldt and Hermann Gehring. A hybrid genetic algorithm for the container loading problem. *European Journal of Operational Research*, 131(1):143–161, 2001.

[6] Andreas Bortfeldt, Hermann Gehring, and Daniel Mack. A parallel tabu search algorithm for solving the container loading problem. *Parallel Computing*, 29(5):641–662, 2003.

[7] Browne C, Powley E, Whitehouse D, Lucas s, and Cowling P.I. A survey of monte carlo tree search methods. *IEEE Transactions on computational intelligence and AI in games*, 4:1 –43, 2012.

[8] A. Weinstein C. Mansley and M. L. Littman. Sample-based planning for continuous action markov decision processes. *Proc. 21st Int. Conf. Automat. Plan. Sched*, pages 335–338, 2011.

[9] G. M. J.-B. Chaslot J.-B. Hoock A. Rimmel O. Teytaud S.-R. Tsai S.-C. Hsu C.-S. Lee, M.-H. Wang and T.-P. Hong. The computational intelligence of mogo revealed in taiwan's computer go tournaments. *IEEE Trans. Comp. Intell. AI Games*, 1:73–89, 2009.

[10] M. Müller C.-S. Lee and O. Teytaud. Guest editorial: Special issue on monte carlo techniques and computer go. *EEE Trans. Comp. Intell. AI Games*, 2:225–228, 2010.

[11] T.-P. Hong-G. M. J.-B. Chaslot J.-B. Hoock A. Rimmel-O. Teytaud C.-S. Lee, M.-H. Wang and Y.-H. Kuo. "a novel ontology for computer go

knowledge management. *Proc.IEEE Int. Conf. Fuzzy Sys*, pages 1056–1061, 2009.

[12] G. M. J.-B. Chaslot, S. Bakkes, I. Szita, and P. Spronck. Monte-carlo tree search: A new framework for game ai. *Proc. Artif. Intell. Interact. Digital Entert. Conf*, pages 216–217, 2008.

[13] T Fanslau and A Bortfeldt. A tree search algorithm for solving the container loading problem. *INFORMS J Comput*, 22(2):222–35, 2010.

[14] Sándor P Fekete, Jörg Schepers, and Jan C Van der Veen. An exact algorithm for higher-dimensional orthogonal packing. *Operations Research*, 55(3):569–587, 2007.

[15] John A George and David F Robinson. A heuristic for packing boxes into a container. *Computers & Operations Research*, 7(3):147–156, 1980.

[16] M. L. Ginsberg. Gib: Imperfect information in a computationally challenging game. *J. Artif. Intell. Res*, 14:303–358, 2001.

[17] José Fernando Gonçalves and Mauricio GC Resende. A parallel multi-population biased random-key genetic algorithm for a container loading problem. *Computers & Operations Research*, 39(2):179–190, 2012.

[18] K He and W Huang. An efficient placement heuristic for three-dimensional rectangular packing. *Comput Oper Res*, 38(1):227–33, 2011.

[19] Leonardo Junqueira, Reinaldo Morabito, and Denise Sato Yamashita. Mip-based approaches for the container loading problem with multi-drop constraints. *Annals of Operations Research*, 199(1):51–75, 2012.

[20] Leonardo Junqueira, Reinaldo Morabito, and Denise Sato Yamashita. Three-dimensional container loading models with cargo stability and load bearing constraints. *Computers & Operations Research*, 39(1):74–85, 2012.

[21] A Lim, B Rodrigues, and Y Wang. A multi-faced buildup algorithm for three-dimensional packing problems. *Omega-Int J Manage S*, 31(6):471–81, 2003.

[22] Jiamin Liu, Yong Yue, Zongran Dong, Carsten Maple, and Malcolm Keech. A novel hybrid tabu search approach to container loading. *Computers & Operations Research*, 38(4):797–807, 2011.

[23] Ana Moura and Jose Fernando Oliveira. A grasp approach to the container-loading problem. *IEEE Intelligent Systems*, 20(4):50–57, 2005.

[24] Banco Mundial. Tráfico marítimo de contenedores (teu: unidades equivalentes a 20 pies). 2018.

[25] H. Nakhost and M. Müller. Monte-carlo exploration for deterministic planning. *Proc. 21st Int. Joint Conf. Artif. Intell*, pages 1766–1771, 2009.

[26] Manfred Padberg. Packing small boxes into a big box. *Mathematical Methods of Operations Research*, 52(1):1–21, 2000.

[27] F Parreño, R Alvarez-Valdes, JF Oliveira, and JM Tamarit. Neighborhood structures for the container loading problem: a vns implementation. *J Heuristics*, 16(1):1–22, 2010.

[28] F Parreño, R Alvarez-Valdes, JM Tamarit, and JF Oliveira. A maximal-space algorithm for the container loading problem. *INFORMS J Comput*, 20(3):412–22, 2008.

[29] A. Sabharwal and H. Samulowitz. Guiding combinatorial optimization with uct. *Proc. 21st Int. Conf. Automat. Plan. Sched*, 2011.

[30] G Scheithauer. Algorithms for the container loading problem. *Oper Res*, pages 445–52, 1992.

[31] B. Sheppard. World-championship-caliber scrabble. *Artif.Intell*, 134:241–275, 2002.

[32] K. Yoshizoe Y. Tanabe and H. Imai. A study on security evaluation methodology for image-based biometrics authentication systems. *Proc. IEEE Conf. Biom.: Theory, Applicat. Sys*, pages 1–6, 2009.

[33] W Zhu and A Lim. A new iterative-doubling greedy-lookahead algorithm for the single container loading problem. *Eur J Oper Res*, 222(3):408–17, 2012.

[34] W Zhu, WC Oon, A Lim, and Y Weng. The six elements to block-building approaches for the single container loading problem. *Appl Intell*, 37(3):1–15, 2012.